Introduction aux Tests logiciels

CogniTIC

Building Together Better IT Solutions

Plan du cours

- Partie 1 : Principes généraux
- Partie 2 : Les types de test
- Partie 3 : Les outils de test

Partie 1

PRINCIPES GÉNÉRAUX

Principes généraux

- Préambule
- Quelques exemples de « Bugs »
- Définitions des tests
- Les catégories de défaut
- Les difficultés liées aux tests
- Les différentes façons de classifier les tests
- Quelques principes
- La stratégie de tests

Préambule

Préambule

- Les tests existent depuis toujours (mauvaise perception)
- Prise de conscience actuelle de la part des sociétés et notamment des directions informatiques (risque de dysfonctionnement, perte financière, retard…)
- Les tests rassurent et permettent de palier aux erreurs humaines
- Le sujet des tests est vaste
- Plan de la présentation
  - Généralités
  - Techniques
  - Outils

Quelques exemples de « bugs »

Quelques exemples de « bugs »

https://www.youtube.com/watch?v=N6PWATvLQCY

Quelques exemples de « bugs »

- **Fin des Années 80 -** Therac 25 - Machine à radiothérapie. Elle envoyait 20 000 rad au lieu de 200 rad. -> Mort de patients
- **1990 -** Défaillance du système télécommunication, 9 heures sans téléphone aux Etats-Unis. Le système a été mis à jour mais n'a pas été testé avant.
- **1991 -** Sleipner A (une plateforme d'exploitation pétrolière) a eu ses fondations qui se sont effondrées qui provoqua un séisme de magnitude 3. Le logiciel de modélisation s'est trompé dans le calcul de l'épaisseur nécessaire pour soutenir la plateforme.

Quelques exemples de « bugs »

- **1992 -** Les ambulances de Londres sont mal orientées par le logiciel. Des pertes humaines sont à déplorer
- **1996 -** Explosion de la fusée Ariane 5 au bout de 30 secondes de vol suite à une erreur de conversion de données numériques
- **1999** - La sonde Mars Climate Orbiter qui devait aller se placer sur l'orbite de Mars pour en étudier le climat ne s'est pas placée sur l'orbite mais est allée se cracher sur Mars. 2 Equipes de développement :
  - **sonde -> unité anglo-saxone**
  - **logiciel de contrôle -> unité métrique**

Quelques exemples de « bugs »

- **2004** - Défaillance du système d'alarme d'une centrale qui produisit une coupure d'électricité aux Etats-Unis et au Canada
- **2006** - Deux grandes banques françaises exécutent un double débit pour plus de 400 000 transactions
- **2010 -** Etats-Unis : Trop de détenus dans les prisons, logiciel pour analyser les dossiers afin de libérer les moins dangereux… Mais le logiciel n'a pas tenu compte de l'entièreté du dossier => Les pires ont été relâchés.

Définition du test logiciel

Définition du test logiciel

- **Selon Glendford.J Myers dans « The art of software testing »:** « Un test réussi n'est pas un test qui n'a pas trouvé de défauts, mais un test qui a effectivement trouvé un défaut. »

- **Selon l'IEEE (Institute of Electrical and Electronics Engineers):** « Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus. »

« On constate une ANOMALIE
due à un DEFAUT du logiciel lui-même
dû à une ERREUR de programmeur. »

Les catégories de défaut

Les catégories de défaut

Les erreurs peuvent être de divers ordres.

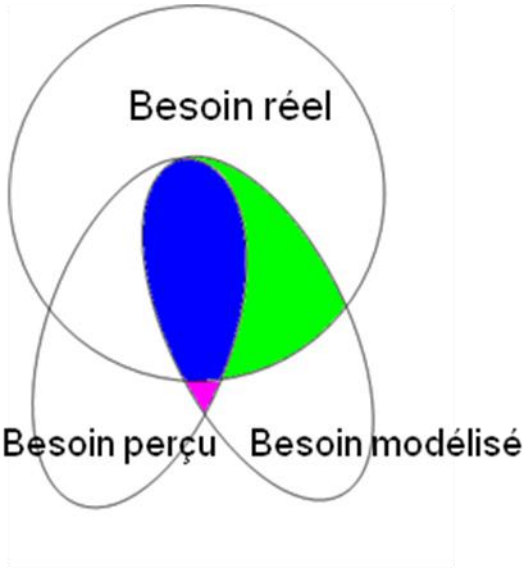
Il peut s'agir d'erreurs :

- de calcul
- de logique
- d'entrée / sortie
- de traitement des données (dépassement de tableau)
- d'interface (communication entre les programmes)
- de définition des données

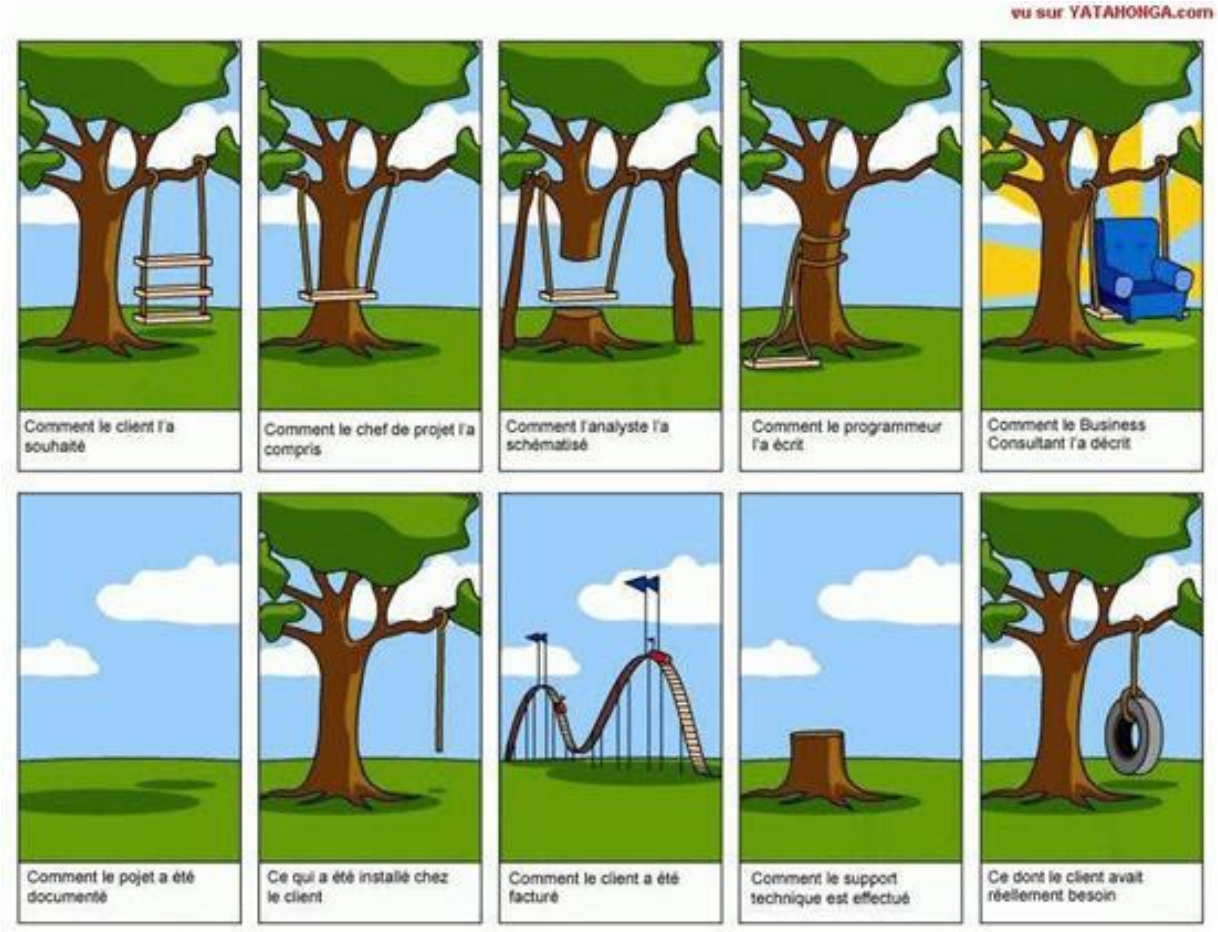
Les difficultés liées aux tests

- Impossible de réaliser un test exhaustif (variables)
- Qualité des tests dépend données utilisées et disponibles
- Impossible de supprimer l'erreur humaine
- Difficultés d'ordre psychologique, culturel
- Manque d'intérêt pour les tests
- Taille et complexité des programmes
- Différence entre environnement de développement et de production
- Perte d'information naturelle

Concernant la perte d'information (1/3)



Concernant la perte d'information (2/3)



Concernant la perte d'information (3/3)

https://www.youtube.com/watch?v=BKorP55Aqvg

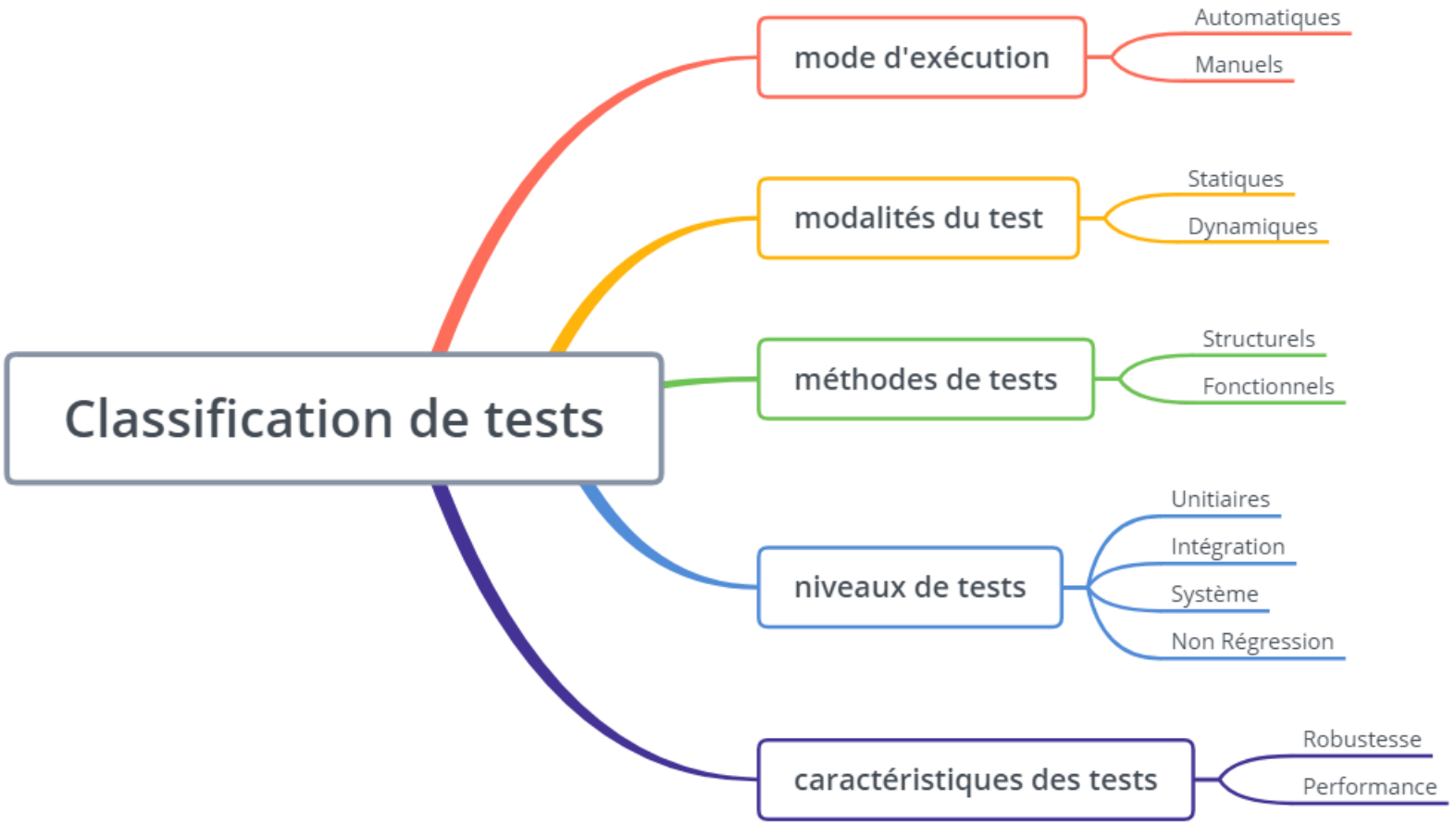
Les différentes façons de classifier les tests

Les différentes façons de classifier les tests

Il n'existe pas de classification officielle.

Chaque ouvrage, chaque auteur, chaque site définit à sa manière les différentes techniques de tests.

Les différentes façons de classifier les tests



Selon le mode d ’exécution

- Tests manuels
  - Les tests sont exécutés par le testeur qui vérifie les traitements… et compare les résultats obtenus avec ceux attendus
  - Ces tests sont **fastidieux** et offrent une plus grande possibilité d’erreurs humaines. Ces tests sont très vite **ingérables** dans le cas d’applications de grandes tailles.
- Tests automatiques
  - Le testeur est en partie déchargé des tests dans la mesure où les tests sont réalisés par des outils (JUnit par exemple dans le monde Java).

Selon les modalités du test

- Tests statiques
  - Les tests sont réalisés «par l'humain» (testeur), sans machine, **par lecture du code** dans le but de trouver des erreurs (revue de code…).
- Tests dynamiques
  - On exécute le système de manière à tester l'ensemble des caractéristiques. **Chaque résultat est comparé aux résultats attendus.**

Selon les méthodes de tests

- Structurels (boîte blanche)
  - Les tests structurels reposent sur des **analyses du code source**
- Fonctionnels (boîte noire)
  - Les tests fonctionnels **reposent sur une spécification** du programme. Le code source du programme n'est pas utilisé. Les tests fonctionnels permettent d'écrire les tests bien avant le « codage »

Exemple

Dans un Word :

1. Module : Fournisseurs

1.1. Fonctionnalité : Rechercher

1.1.1. Scénario : Je souhaite rechercher un fournisseur sur base de …

1.1.1.1. Référence existante

Code	Alternative	Résultat attendu	Résultat obtenu
1.1.1.1.A	F-2020-00001	Valeur retournée	
1.1.1.1.B	F-2020-00004	Pas de valeur	

1.1.1.2. Nom

Code	Alternative	Résultat attendu	Résultat obtenu
1.1.1.2.A	BelPierre	Valeur retournée	
1.1.1.2.B	BalPierre	Pas de valeur	

1.1.1.3. Adresse

Code	Alternative	Résultat attendu	Résultat obtenu
1.1.1.3.A	…	Valeur retournée	
1.1.1.3.B	…	Pas de valeur	

1.1.1.4. Contact

Code	Alternative	Résultat attendu	Résultat obtenu
1.1.1.4.A	…	Valeur retournée	
1.1.1.4.B	…	Pas de valeur	

1.1.1.5. Actif

Code	Alternative	Résultat attendu	Résultat obtenu
1.1.1.5.A	…	Valeur retournée	
1.1.1.5.B	…	Pas de valeur	

1.1.1.6. Combinaison des critères

Code	Alternative	Résultat attendu	Résultat obtenu
1.1.1.6.A	Référence + Nom	Valeur retournée	
1.1.1.6.B		Pas de valeur	
1.1.1.6.C	Référence + Adresse	Valeur retournée	
1.1.1.6.D		Pas de valeur	

…

Exemple

Dans un Excel :

	Module	Fonctionnalité	Scénario	Alternative	Résultat attendu
1.1.1.1.A.	Fournisseurs	Rechercher	Je souhaite rechercher un fournisseur sur base de	Référence existante	Pas valeur
1.1.1.1.B	Fournisseurs				Valeur retournée
1.1.1.2.A.	Fournisseurs			Nom	Pas valeur
1.1.1.2.B	Fournisseurs				Valeur retournée
1.1.1.3.A.	Fournisseurs			Adresse	Pas valeur
1.1.1.3.B	Fournisseurs				Valeur retournée
1.1.1.4.A.	Fournisseurs			Contact	Pas valeur
1.1.1.4.B	Fournisseurs				Valeur retournée
1.1.1.5.A.	Fournisseurs			Actif	Pas valeur
1.1.1.5.B	Fournisseurs				Valeur retournée
1.1.1.6.A	Fournisseurs			Combinaison des critères	Pas valeur
1.1.1.6.B	Fournisseurs				Valeur retournée
1.1.2.1.A	Fournisseurs	Créer un fournisseur	Je souhaite rechercher un fournisseur sur base de	Avec tous les champs	
1.1.2.1.A	Fournisseurs			Avec uniquement les champs requis	
1.1.2.1.A	Fournisseurs			Sans données	
1.1.2.1.A	Fournisseurs				
1.1.3.1.A	Fournisseurs	Gestion des contacts - Création	Je dois pouvoir ajouter un contact		
1.1.4.1.B	Fournisseurs	Gestion des contacts - Modification			
1.1.5.1.B	Fournisseurs	Gestion des contacts - Supprimer			

Selon les niveaux de tests

- **Unitaires**
  - S'assurer que les **composants logiciels pris individuellement** sont conformes à leurs spécifications et prêts à être regroupés
- **D'intégration**
  - S'assurer que **les interfaces des composants sont cohérentes entre elles** et que le résultat de leur intégration permet de réaliser les fonctionnalités prévues
- **Système**
  - S'assurer que **le système complet**, matériel et logiciel, **correspond bien à la définition des besoins** tels qu'ils avaient été exprimés. On parle de validation ou de recette
- **De non régression**
  - **V**érifier que la correction des erreurs n'a **pas affecté les parties déjà développées et testées**. Cela consiste à systématiquement rejouer les tests déjà exécutés.
- Ces tests sont réalisés tout au long du cycle de vie du logiciel

Selon les caractéristiques des tests

- **De Robustesse**
  - Permet d'analyser le système dans le cas où **ses ressources sont saturées** ou bien d'analyser les réponses du système aux sollicitations proches ou hors des limites des domaines de définition des entrées

- **De performance**
  - Permet d'évaluer la capacité du programme à fonctionner correctement vis-à-vis des **critères de flux de données et de temps d'exécution**

Principes

Principes

- Si possible faire tester par un autre développeur
- Ne jamais partir du principe qu'un test ne trouvera pas d'erreurs
- Examiner et mémoriser les rapports de tests
- A la moindre modification, ne pas hésiter à refaire les tests
  - Tests de non régression

La stratégie de tests

Généralités (1/2)

- Une stratégie de tests peut représenter **50% de la charge totale d'un projet**
- La stratégie de test **doit être intégrée** au processus de développement du logiciel

- La stratégie de test dépend :
  - De la **criticité du produit** à réaliser
  - Du **coût** de développement

- Une stratégie consiste à définir :
  - **Les ressources** mises en œuvre (équipes, testeurs, outils…)
  - **Les processus** de test

- Finalement, la stratégie de tests tient compte :
  - Des méthodes de **spécification**, de conception
  - Du **langage** de programmation utilisé
  - Du **type d'application** (temps réel, base de données…)

Généralités (2/2)

- L'activité de test est un PROJET à part entière

- C'est la raison pour laquelle nous retrouvons l'ensemble des caractéristiques d'un projet :
  - Organisation des équipes
  - Planification et contrôle (planification, estimation des charges,…)
  - Analyse et conception
  - Evaluation des risques
  - Gestion des incidents
  - Evaluation et « reporting »
  - Clôture (recette ou arrêt des tests)
  - Bilan projet
  - Amélioration des processus
  - ...

La stratégie

- L'ensemble de la stratégie de tests est détaillé dans le **Plan Qualité Projet** (PQP)

- Le plan qualité projet est très important. Il va notamment :
  - **Définir l'organisation** à mettre en place (équipe de tests : Testeur, CP qui enregistre et analyse les métriques et les incidents, élabore les tableaux de bord)
  - **Définir les responsabilités** et relations entre les différents intervenants
  - **Définir les types et les objectifs** de tests pour chacun des niveaux (tests unitaires, tests d'intégration, tests de validation)
  - **Définir les outils** qui seront utilisés
  - **Définir les moyens et les délais** à investir dans l'activité de tests

Plan de test

- Il existe autant de **plan de tests** que de phases de qualification du produit :
  - Au dossier de SPECIFICATION correspond le plan de tests de VALIDATION
  - Au dossier de CONCEPTION GENERALE correspond le plan de tests d'INTEGRATION
  - Au dossier de CONCEPTION DETAILLEE correspond le plan de tests UNITAIRES

- L'objectif de chaque plan de tests est de fournir un programme pour vérifier que le logiciel produit satisfait les spécifications et la conception du logiciel

- Un plan de test doit :
  - **Définir les éléments à tester** et l'ordre dans lequel ils doivent être testés (planifier)
  - **Décrire l'environnement** de tests
  - **Définir la façon dont les tests vont être menés** (procédures répétables : processus exacts à mener, l'historisation, la traçabilité, le reporting, le suivi, le contrôle)
  - Décrire et constituer les **fiches de tests**. L'ensemble des fiches de tests constitue le **dossier de tests**
  - **Fixer les critères d'arrêt** des tests

Rapport de test

- Ensemble des comptes rendus de rapport de tests
- Pour chaque phase de test (unitaires, d'intégration, de validation), l'équipe dédiée aux tests doit élaborer un rapport de tests
- Ce rapport est la synthèse des actions menées suivantes :
  - Exécution des fiches de tests (effectuer les actions décrites)
  - Analyser les résultats obtenus
  - Emettre des fiches de non-conformité si nécessaire

Qualification

- Validation
  - La validation est essentielle. Elle permet de conclure et d'émettre un avis sur le produit développé et sa mise en production : adéquation entre produit et spécifications fonctionnelles et techniques

Partie 2

LES TYPES DE TEST

Partie 2 : Les types de test

- Tests et Cycles de vie
- Les tests UNITAIRES
- Les Tests d'INTEGRATION
- Les tests de CHARGE
- Les tests de VALIDATION

Tests et cycles de vie

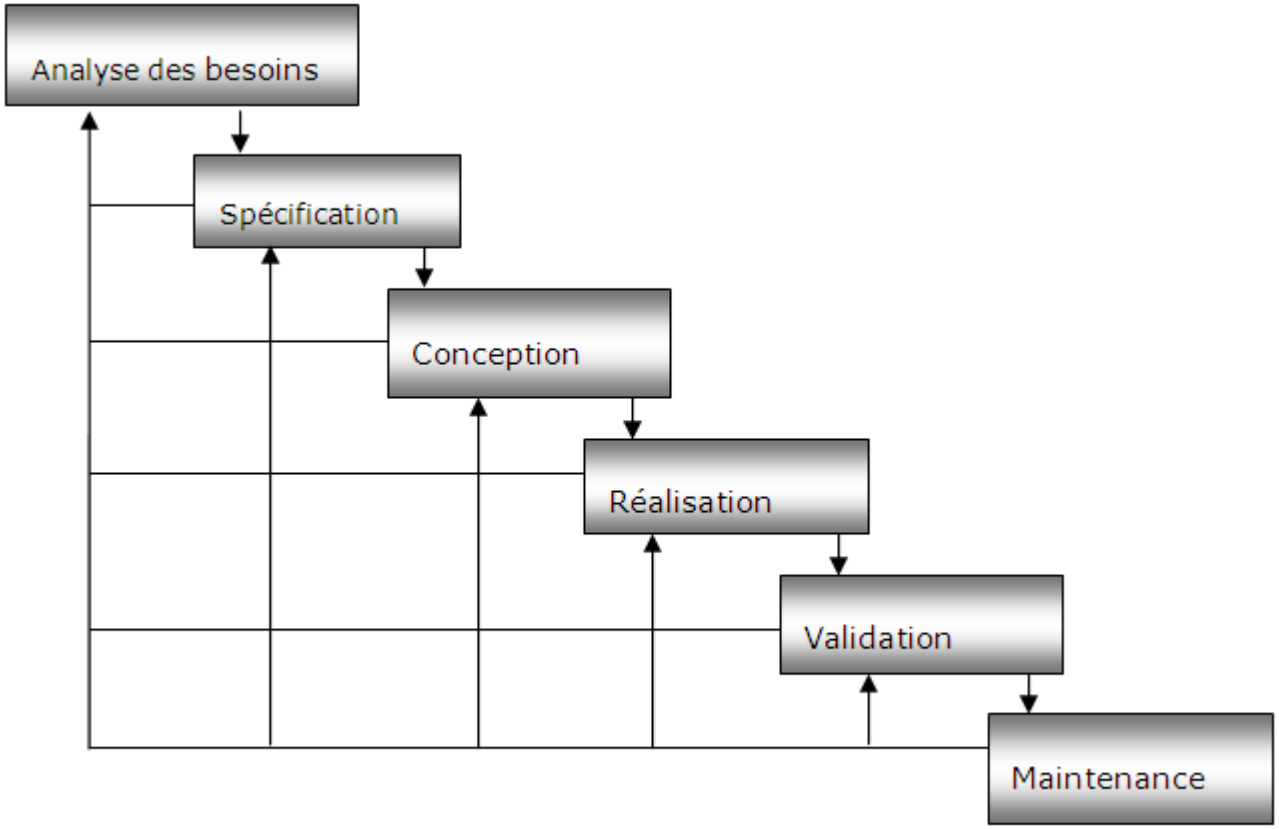
Tests et cycles de vie

- 3 cycles de vie applicables
  - Modèle en cascade
  - Modèle en V
  - Modèle en spirale

Cycle en cascade (1/2)

- Définit les phases séquentiellement
- A la fin de chaque phase, des documents sont créés pour en vérifier la conformité
- Si c'est correct, on passe à la phase suivante
- Si ce n'est pas correct, on retourne en arrière et on envoie un feedback

Cycle en cascade (2/2)

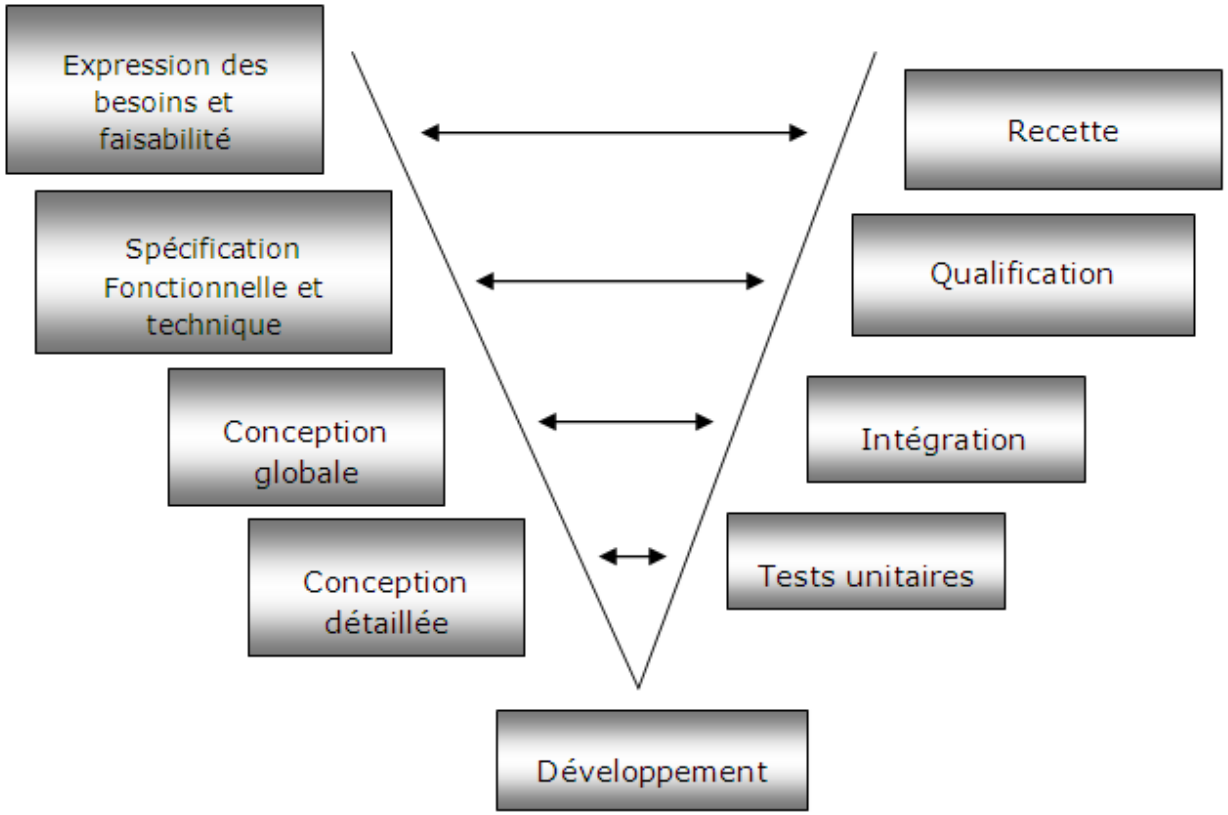


Source : Cours CNAM Génie Logiciel

Cycle en V (1/2)

- Créé suite au manque de réactivité du modèle en cascade. Il part du principe que les procédures de vérification de la conformité doivent être élaborées dès les phases de **conception**
- Le client connait son besoin dans le détail
- Standard depuis les années 1980

Cycle en V (2/2)



Source : adaptée de http://fr.wikipedia.org/wiki/Cycle_de_développement

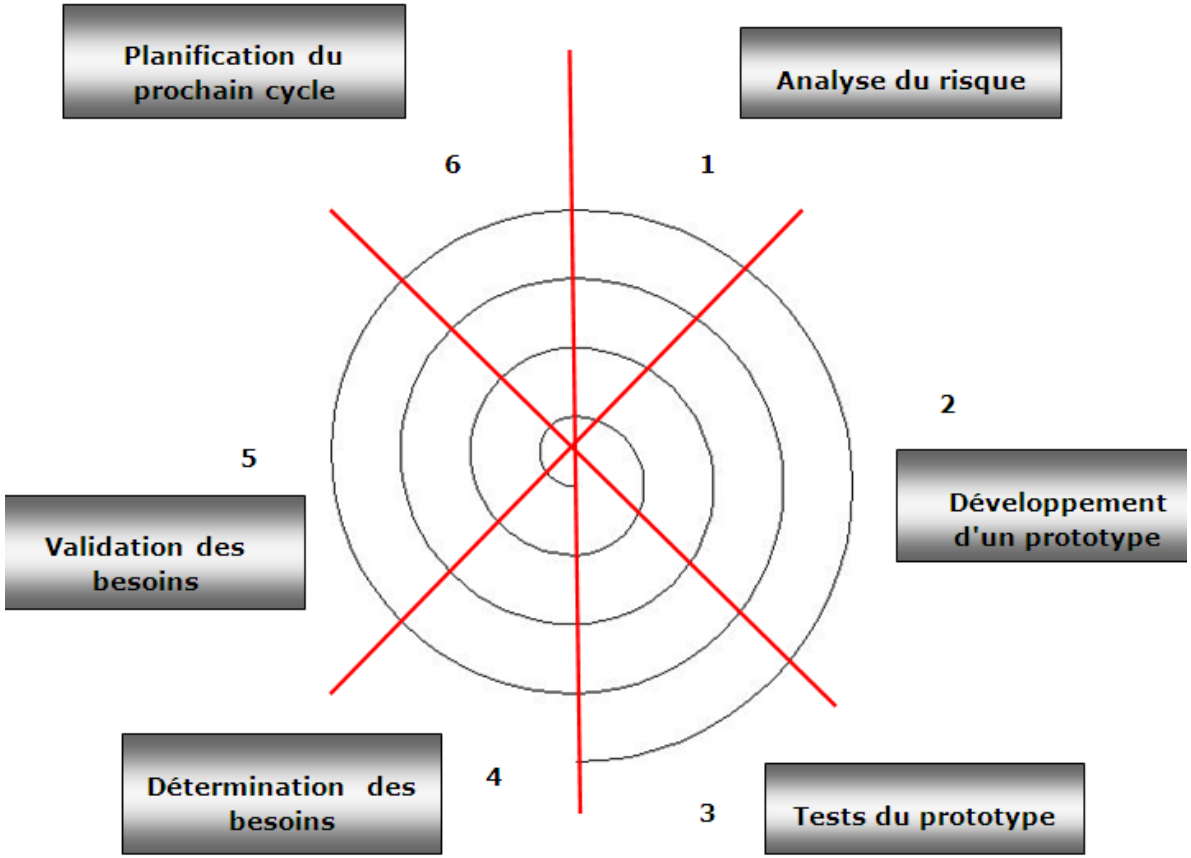
Cycle en spirale (1/3)

- Il est basé sur une approche et une évaluation des **risques**. Un risque identifié est assigné d'une **priorité**. Un prototype possède son cycle de vie propre
- Le cycle en spirale couvre l'ensemble du cycle de vie de développement mais ajoute une dimension managériale et donc non technique

Cycle en spirale (2/3)

- Chaque cycle de la spirale comprend 6 phases :
  - Analyse du risque
  - Développement d'un prototype
  - Test du prototype
  - Détermination des besoins
  - Validation des besoins
  - Planification du prochain cycle

Cycle en spirale (3/3)



Source : adaptée de http://fr.wikipedia.org/wiki/Cycle_de_développement

Conclusion sur les cycles de vie

- Chaque cycle de vie contient les mêmes grandes phases du projet :
  - Analyse du produit
  - Phase de conception
  - Phase d'intégration
  - Phase de validation

Les tests unitaires

Tests unitaires - définition

- Vient du fait qu'une **partie du code** est appelé **Unit**
- Ce type de test va **vérifier un module** du code et son fonctionnement de manière **indépendante** du reste
- Respect aussi des spécifications fonctionnelles
- Ils peuvent être manuels ou automatisés par des logiciels

Tests unitaires - formalisme

- On va créer une fiche de test unitaire qui contient une liste (ou aide mémoire) pour les grandes actions
- Elle permet de préparer les tests
- Chaque analyste ou chef de projets doit la remplir afin d'assurer un passage en recette dans les meilleures conditions
- Donne une garantie de qualité lors du développement (ce n'est pas une contrainte mais un outil !)

Tests unitaires – besoins

- Pour réaliser des tests unitaires, il nous faut :
  - Des jeux de données (fictives, de production, anciens jeux de tests)
  - Des ressources (documents de spécifications, scénarios, fiches de tests, tests précédents)
  - Une démarche

Analyse dynamique structurelle

- Consiste à vérifier la structure du code ainsi que les variables
- On parcourt tous les nœuds, les arcs, et chemins du programme
- On peut vérifier ainsi si un test If Then Else n’est pas utilisé

Analyse dynamique fonctionnelle

- Consiste à vérifier le service rendu (la fonction) mais pas comment il est rendu
- On valide les règles de gestion
- La difficulté réside dans le choix des données de tests pour obtenir les résultats attendus

A vous de tester

- Rendez-vous sur le lien suivant :
- http://test-logiciel.bstorm.be/
- Et faites différents tests pour vérifier que le produit fonctionne parfaitement.

A vous d'écrire les tests

- Exercices du Triangle de Weinberg - Myers
  - Vous devez réaliser les spécifications complètes d'une application. L'objectif de l'application est de considérer un encodage de texte libre comme les côtés d'un triangle. Voici les restrictions émises par le client :
  - La taille maximale de ces côtés est de 9 inclus.
  - La lecture se fait dans la console, sous le format (x,y,z) ou (x, y, z)
  - Le programme doit ensuite déterminer si le triangle est scalène, isocèle ou équilatéral.
  - Vous devez identifier la liste des erreurs possibles liés à l'implémentation de cette méthode afin d'attirer l'attention sur ces points clefs dans le cahier de charge.
  - Pour chaque type d'erreur identifié, il est demande de fournir un exemple d'encodage incorrect afin de permettre au développeur de tester la robustesse de l'application.

Les tests d’intégration

Définition

- On **regroupe** chaque partie testée unitairement afin d'établir une nouvelle version du produit
- Le test d'intégration a pour but de tester que **tout fonctionne ensemble**. Il existe une intégration incrémentale et une intégration globale

Méthodes

- Il existe plusieurs méthodes d'intégration :
  - Big bang
  - Top Down (haut en bas)
  - Bottom up (ascendante)
  - Mixte

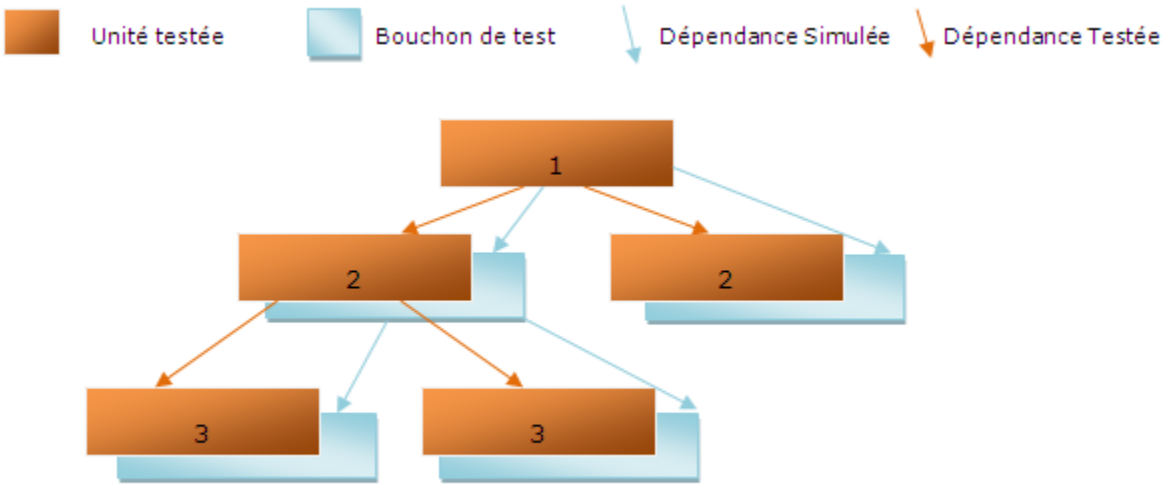
Big Bang

- Le principe est d'intégrer **tous** les composants en une seule étape.
- L'intégration est rapide mais valable que pour les petits projets
- Trop de risques pour les gros projets

Top Down (1/2)

- On déroule le programme de haut en bas. Les modules s'empilent les uns aux autres. Des bouchons simulent les traitements. Les modules doivent être les plus petits possibles
  - Avantages
    - Détection précoce des défauts d'architecture
    - Facilité de compréhension
  - Inconvénients
    - Créer des bouchons prend du temps
    - Effort de simulation des composants absents (risque d'erreurs)
    - Tests tardifs des couches de bases

Top Down (2/2)

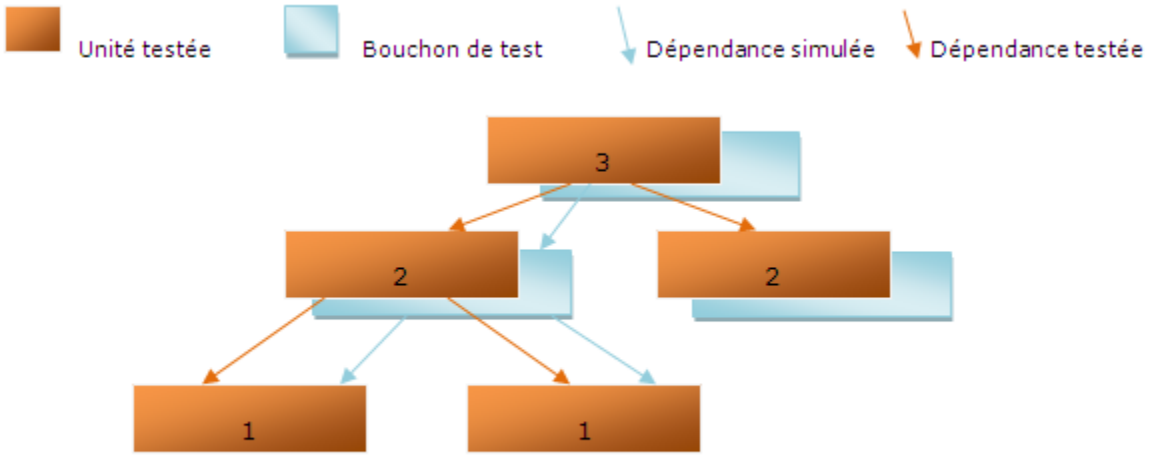


Source : Réalisé par Eric LELEU pour illustration

Bottom Up (1/2)

- On déroule le programme de bas en haut. Les modules fondamentaux sont testés en premier. Des bouchons simulent les traitements. Il est nécessaire de définir les fonctionnalités indispensables et prioritaires aux autres
  - Avantages
    - Faible effort de simulation
    - Définition de jeux d'essais plus facile
    - Fonctionnalités basses plus souvent testées
  - Inconvénients
    - Détection tardive des erreurs majeures

Bottom Up (2/2)



Source : Réalisé par Eric LELEU pour illustration

Intégration mixte

- C’est une combinaison des approches Bottom up et Top down. On l’appelle de temps en temps **boîte grise**
  - Avantages
    - Planning de développement qui gère les composants dans l’ordre de création
    - Les composants les plus critiques sont intégrés en premier
  - Inconvénients
    - Difficulté d’intégration due à la mixité des méthodes

Exemple

- Rendez-vous sur le lien suivant :

- http://exemple.bstorm.be/

- login : sa
- mot de passe : Test1234=

Les tests de charge

Définition

- On expose l'application à des conditions d'exploitations pour valider le système
- Objectifs
  - Tester la performance
  - Maintien des fonctionnalités sur une montée en charge
  - Fiabilité (plateforme, BDD…)

Types de tests de charge

- Principaux types de tests de charge
  - Test de performance
  - Test de dégradation des transactions
  - Test de stress
  - Test d'endurance, de robustesse, de fiabilité
  - Test de capacité, de montée en charge

Planning

- Quand mener les tests de charge ?
  - Le plus tôt possible dans le processus de développement afin de mettre en évidence rapidement les défauts d'infrastructure.

Tests de validation

Objectifs

- Vérifier que toutes les exigences du cahier des charges sont respectées. Ils ont lieu immédiatement après les tests d'intégration
- 2 approches
  - Identifier et tester toutes les fonctions du logiciel
  - Tester les caractéristiques du logiciel (interfaces…)

Exemple

- A l'aide du cahier des charges et de l'analyse, je vous propose un test de validation statique.
- Vous avez à votre disposition une maquette de l'application. Validez ou non que c'est bien de résultat souhaité.

Partie 3

LES OUTILS DE TEST

Partie 3 : Les outils de test

- Les outils de tests
- Le classement
- Les outils d’aide à la réalisation
- Les outils de campagne de tests
- Les outils de tests fonctionnels
- Les outils de tests structurels
- Les outils de tests de performance

Les outils de tests

Pourquoi un outil de test

- L'exemple du triangle : en entrée 3 entiers et en sortie triangle isocèle, scalène ou équilatéral.
  - 14 cas de tests : GJ Myers –«The Art of Software Testing» - 1979
  - Résultats sur des développeurs expérimentés 7,8 en moyenne sur 14.
- Coût d'un test est de 30% à 60% du coût de développement

- L'environnement du SI est toujours plus complexe
- La programmation évolue et implique plusieurs langages
- Détection des bugs plus difficile, débuggeur difficile à mettre en œuvre
- L'exigence qualité du client → SI ou fournisseurs engagement qualité
- Base de la réponse : ISO20000, CMMI, ITIL
- Le SI, les fournisseurs cherchent une réponse pour automatiser les tests

Le classement

Classement

- Classés en 5 groupes d'outils :
  - Les outils d'aide à la réalisation des tests
  - Les outils de campagnes de tests
  - Les outils de tests fonctionnels (boite noire)
  - Les outils de tests structurels (boite blanche)
  - Les outils de tests de performance

Les outils d’aide à la réalisation

Les outils d'aide à la réalisation (1/2)

- Les fonctionnalités communes
  - Capture et ré-exécution des scripts réalisés via une IHM
  - Sauvegarde des tests et des résultats associés
  - Génération de scripts de tests en fonction des langages et des plateformes

Les outils d’aide à la réalisation (2/2)

- Outil de test de fonctionnalités, de régression et de configuration polyvalent
- Automatisation aisée des tests manuels
- Détection du plus grand nombre d'erreurs
- Scénarios de tests adaptés aux objets communs (menus, listes) et spécialisés aux objets spécifiques (environnement de développement)
- Intégration d'un outil de gestion des tests, suivi des erreurs : Team Unifying Platform

Les outils de campagne de test

Description

- Les principales fonctionnalités de ce type d'outils sont
  - La définition de campagnes de tests
  - L'historisation des résultats
  - La gestion des tests de non régression
  - Interfaçage avec tous les outils. Ne sont pas des automates de tests
  - Suites Intégrées d'outils de gestion de tests et automates → éviter la concurrence

Les outils de tests fonctionnels

Définition

- Vérifier la conformité du fonctionnement d'un système vis-à-vis du cahier des charges
- Ces outils évitent la création "manuelle" de scripts de tests
- Offre des fonctionnalités automatiques (génération de scripts de tests, de rapports des résultats attendus et atteints ou non) et des indicateurs graphiques
- Ces outils de génération automatique de tests nécessitent une modélisation fonctionnelle
- Ces outils sont une valeur ajoutée pour les projets de grande taille

Les outils de tests structurels

Définition

- Ces outils permettent de valider ce que fait le logiciel testé
- Ils sont donc complémentaires aux outils de tests fonctionnels qui vérifient, eux, ce que doit faire le logiciel
- C'est pourquoi des éditeurs ont créé des suites comprenant ces deux types de tests

Les outils de tests de performance

Définition

- Les tests de performances sont menés pour des sites Web ou Intranet
- Répondre aux exigences de performance d’accès au site
- Les outils de tests de performance proposent souvent
  - Le test de montée en charge
  - La simulation d'un environnement spécifique
  - L’évolution agressive de l'accès aux ressources

CONCLUSIONS

Conclusions (1/2)

- Les tests ont évolué au fil des années :
  - De 1960 à 1980, il s'agissait d'une **mise au point**.
  - De 1980 à 1990, les tests cherchaient uniquement à **trouver des erreurs**.
  - Depuis 1990, les tests se veulent **préventifs**.

- Les applications sont de plus en plus complexes, les volumes de données sont de plus en plus grands… Il était obligé que les tests occupent à nouveau le **devant de la scène** (Nombreux outils, Certaines sociétés ont leur propre structure de tests…)
- L'Extreme programming (Méthode agile de gestion de projets) qui a compris et intégré les tests dans son cycle itératif de développement
- Même le Cloud Computing, qui est un concept émergent est impacté par ce nouvel élan pour les tests (nuages de serveurs) : on parle déjà de HaaS (Human as a Service) qui consiste à externaliser le **capital humain** ! Autrement dit, les tests !

Conclusions (2/2)

- Nous avons tenté de présenter le plus simplement possible ce vaste sujet des tests
- Il n'existe pas de techniques meilleures que d'autres : Tout dépend de nos besoins, de nos objectifs… Rien n'empêche de combiner plusieurs techniques. C'est ce que nous préconisons d'ailleurs
- Nous avons vu qu'aujourd'hui que les tests faisaient l'objet d'une pratique encore trop souvent artisanale mais que dans un futur proche, les tests seront une activité rigoureuse fondée sur **des modèles et des théories** et qu'elle sera de plus en plus **automatisée**

Il ne fait aucun doute que la politique de tests est aujourd'hui une dimension incontournable de la gestion de projet.